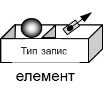
Георги Филев F104081

Домашно 7А. Динамично отреждане на памет, указатели. Образуване на Стек.

Oзначение 1. Указател.

Oзначение 2. Елемент на структура

Означение 3. Променлива – указател.

Ivanх

Означение 4. Нулев адрес.

1. Като използвате графични означения подобни на примерните, дадени горе, посочете графично и с текст (на български език) какво прави всеки от следните оператори:

|  |  |  |
| --- | --- | --- |
| ***оператори*** | ***графична изображение*** | ***текст*** |
| **typedef** **struct** Element \* po; |  |  |
| **struc**t Element { |  |  |
| **int** Data; po Next; |  |  |
| }; |  |  |
|  |  |  |
| **po q;** |  | Инициализираме указател от тип po с име q |
|  |  |  |
| **po p;** |  | Инициализираме указател от тип po с име p |
|  |  |  |
| **q=NULL;** |  | Указателят q няма стойност и не сочи към нищо |
| **и** |  |  |
| **p= New Element;** |  | Правим Р от тип нов елемент |
|  |  |  |
| **p-->data = 78;** |  | Стойността на мястото към което сочи указателя е 78 |
|  |  |  |
| **p-->Next = q;** | 78 | Указателя на елемнта р сочи към q |
|  |  |  |
| **q = p;** |  | Елементът q става равен на p ,тоест сочат към едно и също място |
|  |  |  |
| **p= New Element;** |  | Създаваме нов елемент с име p |
|  |  |  |
| **p-->data = 38;** | 38 | Датата на р е равна на 38 |
|  |  |  |
| **p-->Next = q;** | q | Указателят на елемента п сочи към q |
|  |  |  |
| **q = p;** |  | Q е равно на р |
|  |  |  |
| **p = p-->Next ;** |  | Р е равно на указателят на р |

1. Напишете в кода долу смислени коментари на екран, на български, на мястото на дадените в червено (очевидно, трабва първо да разберете какво прави кодът):

#include<iostream>

#include<clocale>

using namespace std;

typedef struct Element \* po;

struct Element{

int Pole1;

float Pole2;

char Pole3;

};

void main() {

setlocale(LC\_ALL, "Bulgarian");

po p;

p = new  Element;

cout << "създаваме нов елемент от тип Елемент" << endl;

cin >> p->Pole1;

cout << "въвеждаме стойност за Поле1 което е int" << endl;

cin >> p->Pole2;

cout << " въвеждаме стойност за Поле2 което е float" << endl;

cin >> p->Pole3;

cout << " въвеждаме стойност за Поле3 което е char << endl;

cout << p->Pole1 << " " << p->Pole2 << " " << p->Pole3 << endl;

cout << endl;

po q;

q = new Element;

cout << " създаваме нов елемент от тип Елемент с име q" << endl;

cin >> p->Pole1;

cout << " въвеждаме стойност за Поле1 което е int" " << endl;

cin >> p->Pole2;

cout << " въвеждаме стойност за Поле2 което е float" " << endl;

cin >> p->Pole3;

cout << " въвеждаме стойност за Поле3 което е char" " << endl;

cout << p->Pole1 << " " << p->Pole2 << " " << p->Pole3 << endl;

cout << "изписваме всички стойноси на елемента р" << endl;

q = p;

cout << endl;

cout << "стойността на Element q е равна на р" << endl;

cout << p->Pole1 << " " << p->Pole2 << " " << p->Pole3 << endl;

cout << " изписваме всички стойноси на елемента р " << endl;

cout << p->Pole1 << " " << p->Pole2 << " " << p->Pole3 << endl;

p->Pole1 = int(p->Pole2);

cout << "Поле1 от елемента р е равно на Поле2 от елемента р като е typecast-нато като int" << endl;

cout << p->Pole1 << " " << p->Pole2 << " " << p->Pole3 << endl;

}

p

na

1. Като използвате графични означения подобни на примерните горе, нарисувайте какво прави **всеки** от операторите при две преминавания през първия и после – през втория цикъл:

|  |  |  |
| --- | --- | --- |
| ***оператори*** | ***графична изображение*** |  |
| ***struct*** Element { |  |  |
| ***int*** Data; po Next; |  |  |
| }; |  |  |
| ***void*** ***main***() |  |  |
| {po na; po p; |  |  |
|  |  |  |
| **na = NULL;** |  |  |
| **while (потребителят иска да въведе още данни)** | ***графична изображение първо преминаване*** | ***графична изображение второ преминаване*** |
|  |  |  |
| **p = new Еlement;** |  | Вторият път това което се случва е че сегашният елемент сочи към предишния |
|  |  |  |
| **p->data = вход от клавиатура** | Указателят P сочи към na |  |
|  |  |  |
| **p->next = na;** | Na= p |  |
|  |  |  |
| **na = p;** |  |  |
|  |  |  |
| ***край на while*** |  |  |
|  | ***графична изображение*** |  |
|  |  |  |
| **p = na;** |  |  |
| **while (*p* не сочи нулев адрес)** | ***графична изображение първо преминаване*** | ***графична изображение второ преминаване*** |
|  |  |  |
| **печат на** ***p->data*** |  |  |
|  |  |  |
| **p = p->next;** |  |  |
|  |  |  |
| *край на while* | Указателят на р сочи към предишния |  |
| } |  |  |

1. Въпрос, зададен на тест по алгоритми и структури, е:

***Обяснете в свободен текст кога рекурсивният алгоритъм quick sort работи със сложност точно n\*log(n)?***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ПОСОЧЕТЕ ЗА ВСЕКИ ОТГОВОР ИМА ЛИ, и ако има - КОЕ Е: | | |  |
| ***Представен отговор*** | Неточен или неясен термин(и) | Невярно или неясно твърдение | Важно допълнение към текста, за да стане верен | Напишете тук ваш коментар |
| QuickSort рьботи със сложност n\*log(n), когато избраният еталон е точно средната по големина стойност в масива. |  |  | ДА |  |
| В най-добрия случай, когато еталонът е средният по големина елемент. |  | ДА |  |  |
| когато се изполнява рекурсия и почва да се дели масивът всеки път на 2 и за двете части (при търсенете на нов еталон) | да |  |  |  |
| Когато масивът е удобен да си го разделяме на по 2 равни части и с правилен еталон. |  | ДА |  |  |
| Най-лошият случай е, когато една от двете половини въранти се пада с размер n-1. n\*log(n) имаме, когато и двете половинки са равни части или поне близо до това да са равни на брой. | ДА |  |  |  |
| Когато елементът е избран за 'еталон', се пада и среден по големина. Тогава се получават две половини с равна дължина. |  |  | ДА |  |
| Quick sort ще работи със сложност от n\*log(n) когато всеки път при разделяне на дялове бъде избран за еталон точно средният елемент, т.е. когато разделяме масива на две половини. Това ще бъде и най-добрият случай за quick sort. |  |  | ДА |  |
| Всяка операция за разделяне отнема O(n), но всяко делене прави масива два пъти по-малък, следователно имаме log(n) операции. Обобщено това води до: O(n \* log(n)) |  | неясно | Нужно да се спомене |  |
| Сложността е n\*log(n), когато разглеждаме възможно най-добрия случай и се случва, когато разделянето води до равни подмасиви. |  |  | ДА |  |
| Когато за еталон е взето средното число на масива. | ДА |  |  |  |
| Алгоритъмът работи със сложност n\*log(n) в най-добрият си случaй, когато за еталон винаги взимаме средната стойност за разглеждания дял. |  | ДА |  |  |
| n\* log(n) е най - благоприятният случай на рекурсивният алгоритъм quick sort.  Това се получава когато за всеки елемент позицията на дяла се намира в средата, което води до балансирано дърво с височина log (n), следователно общата стойност става. O(n\*log n). Казано с други думи еталонът винаги трябва да е средният елемент. |  |  | ДА |  |
| Когато еталонът е винаги така избран и данните са така наредени, че разделянето на масивът при потъване е винаги на две равни части. Пример: при наредени данни. |  |  | ДА |  |

В таблицата има и отговори, които са коректни и пълни.

ДОМАШНОТО СЕ ПРЕДАВА С ТОЧКИТЕ НА ЗАДАНИЕТО!